A Novel Architecture for High Quality AWGN Generation Based on Central Limit Theorem

by

Muhammad Jameel Nawaz Malik

This thesis is submitted in partial fulfillment of the requirements for the degree of Masters of Science in Electrical Engineering (MS EE)

School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

May 2011

APPROVAL

It is certified that the contents of thesis document titled, "A Novel Architecture of High Quality AWGN Generation Based on Central Limit Theorem " submitted by Mr. Muhammad Jameel Nawaz Malik have been found satisfactory for the requirement of degree.

Advisor: Dr. N. D. Gohar

Signature: _____

Date: _____

Committee Member1: Mr. Jamshaid Malik

Signature: _____

Date: _____

Committee Member2: Dr. Khurram Aziz

Signature: _____

Date: 10-May-2011

Committee Member3: Dr. Osman Hasan

Signature: _____

Date: _____

TO MY LOVING FAMILY

CERTIFICATE OF ORIGINALITY

I declare that the research work titled **"A Novel Architecture for High Quality AWGN Generation Based on Central Limit Theorem"** is my own work and to the best of my knowledge. It contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at SEECS or any other education institute, except where due acknowledgment, is made in the thesis. Any contribution made to the research by others, with whom I have worked at SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic is acknowledged. I also verified the originality of contents through plagiarism software.

Author Name: Muhammad Jameel Nawaz Malik

Signature: _____

ACKNOWLEGMENTS

I am grateful to Almighty Allah who gave me courage and support to complete this thesis. I owe my deepest gratitude to my thesis advisor, Dr. N. D. Gohar, for his kind attention and guidance during this thesis. I am, also, highly obliged and grateful to my thesis co-advisor Mr. Jamshaid Sarwar Malik, for his valuable guidance throughout my research work. I am, also, grateful to Dr. Syed Ali Khayyam and Mr. Moin-ud-Din for their valuable guidance during the analysis phase of this research work. I am, also, thankful to my worthy Committee members, Dr. Osman Hasan and Dr. Khurram Aziz, for their support and becoming a part this work. I am, also, thankful to professional members of CEFAR lab for always supporting and encouraging me. I am extremely gratified to ICT R&D Fund PTCL, Pakistan for their breed financial support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Gaussian Random Numbers (GRNs) are required for simulations in a wide variety of applications. For example, channel code evaluation, simulation of economic systems and product failure simulations, etc. Mostly, simulations are carried out using systems based on digital signal processor or other software programmable devices. Such systems generate GRNs using software libraries to evaluate complex trigonometric functions like natural logarithm, exponential functions, etc. However, optimized hardware implementation of GRNs generator can operate many times faster than optimized software implementations.

Hardware implementation of GRNs generator generally involves transformation of uniformly distributed random numbers and has always been a challenging task. Central Limit Theorem (CLT), although very simple to implement, has never been used to generate high quality GRNs. This is because the direct implementation of CLT provides very poor accuracy in the tail region of probability density function (PDF). This work achieves high quality GRNs generator. The empirical model of the error in CLT is compensated through deployment of a low complexity compensation block. A novel non-uniform segmentation algorithm is presented for degree one piecewise polynomial approximation to non-linear error function. We have proposed a novel architecture of GRNs generator which requires only 420 configurable slices and 01 DSP block of Xilinx Virtex-4 XC4VLX15 operating at 220 MHz. The architecture achieves high tail accuracy of 6σ and is scalable to achieve even higher accuracy with minimal increase in hardware resources. The accuracy of GRNs generator is validated using statistical goodness of fit tests.

CHAPTER 1

INTRODUCTION

1.1 Motivation

A fast, compact and high quality Gaussian random number generation is a key capability of simulations across a wide range of disciplines. For example, Channel code evaluation, Monte-Carlo (MC) simulations, financial modeling, molecular dynamics simulation and product failure simulations, etc. Most of the processes in nature are Gaussian distributed since they tend to be a balanced sum of many unobserved random events and by virtue of Central Limit Theorem (CLT) [1], sum of sufficiently large random numbers tend to become Gaussian distributed. In simulations, where probability of occurrence of an event is very low, high tail accuracy Gaussian Random Numbers (GRNs) are required.

High tail accuracy GRNs generators are of significance assistance in characterization of very low BER systems (high Signal-to-Noise (SNR) ratio) [2]. One such example is BER simulation of wireless radio standards. Some of which have maximum allowable BER of less than 10^{-10} for specified SNRs. Other important examples are turbo codes and Low-Density Parity-Check (LDPC) codes which are currently the focus of an intense research due to their ability to approach Shannon bound very closely and with only moderate decoding complexity [12]. Simulating occurrence of such events, requires a tail accuracy of at least 6σ [2]. It takes about 2.5 hours to generate 10^9 Gaussian samples on a dual core Pentium processor using an optimized software simulator written in C [2]. This means that it would take about 27000 years to generate 10^{17} Gaussian samples. Clearly, software simulations cannot provide an adequate solution to examine the behavior of such codes at very low BER rates [12].

Recent advances in Field Programmable Gate Array (FPGA) technology are providing cost effective, fast and compact solutions towards hardware implementations of algorithms. Moreover, the user programmability of FPGA facilitates debugging capability and design characterization, which can reduce the total design time significantly [2]. Hence, this work is strongly motivated by the efficient hardware Gaussian random generation which should be high quality, fast, compact and reliable.

In last two decades, intense research has been reported regarding the efficient implementation of GRNs in hardware. Simulations are usually done in software e.g. Digital Signal Processors (DSPs) and other software programmable devices. To generate GRNs, such devices use software libraries to evaluate complex functions e.g. natural logarithms, square root, exponential functions etc. These complex functions are now evaluated in hardware using piecewise polynomial approximation techniques. But, hardware implementation of these functions requires lot of computational resources and power consumption. Hence, there is a need to come up with a simple, fast and cost effective solution towards the implementation of GRNs in hardware.

This work presents an efficient hardware implementation of high quality GRNs generator based on Central Limit Theorem (CLT). CLT, due to its inherent Gaussian like characteristics, can be an efficient method for generating high quality GRNs. After a detailed empirical data analysis, error models for CLT have been computed. The error distributions are compensated using a novel non-uniform segmentation algorithm. Tail accuracy of 6σ is guaranteed which is considered well enough for all practical purposes. The proposed GRNs generator is scalable to achieve even higher accuracy with minimal increase in hardware resources.

1.2 Thesis Contributions

Empirical error model based on the actual data samples is computed for a specified value of *n* (number of additions) in CLT. A detailed analysis of error model is provided using appropriate bit width of data path and number of additions (*n*) for a target sigma or standard deviation value. A novel non-uniform segmentation algorithm is introduced that computes coefficients for first degree piecewise polynomial approximation of error functions. A novel architecture of CLT based GRNs generator is presented that produces high tail accuracy GRNs at minimal hardware cost.

1.3 Thesis Organization

There are five chapters in this thesis organized as follows:

In Chapter 2, an introduction to GRNs generators is given. After a brief history, various algorithms used for generating Additive White Gaussian Noise (AWGN) are described in detail including the work related to Central Limit Theorem.

In Chapter 3, empirical error models for Central Limit Theorem are computed. Compensation algorithm is, also, described in detail using a novel segmentation algorithm.

In Chapter 4, the simulation and synthesis results are given and explained in detail. Also, comparison with previous methods is given and discussed.

In Chapter 5, summarizes the work and concludes our thesis with proposals of some possible extensions to this work.

In the end, references of the work done by different people in this field are given.

CHAPTER 2

LITERATURE SURVEY

2.1 Gaussian Random Number Generators (GRNGs)

Normally distributed random numbers are referred to as Additive White Gaussian Noise (AWGN). A variety of algorithms have been reported in literature to generate Gaussian Random Numbers (GRNs) with a varying degree of computational complexity and accuracy [3]. A popular method is to convert uniformly distributed random numbers to Gaussian by transformation algorithms [2]. These transformation algorithms have been implemented both in software [3] and hardware [1]. Software solutions are used in applications where primary concern is cost and flexibility. However, in applications where high values are required, software GRN generators fail to provide desired number of samples in short time. For example, at least 10^{12} samples are required to detect occurrence of an event in 6σ region. Fastest available machines today produce around 1 to 10 million GRNs per second [1]. At this rate, such a machine will take more than a day to generate desired 10^{12} samples. This situation worsens by order of magnitudes for simulation of higher tail accuracy. Another extreme example, where software GRNs generators fail is real time radio channel emulation. GRNs generators are an essential part of any radio channel model [4]. Some complicated fading channel models [4] require large samples of GRNs. For real-time emulation, these channel models have to be executed at tens of Mega Hertz (symbol rate of underlying wireless standard). Clearly software GRN generators don't have the capacity to execute such models in real-time. Hence, efficient hardware GRNs generators are the only option for such applications.

Figure 2.1, shows the basic methodology to generate GRNs. First step is to produce base numbers from a uniform distribution using Linear Feedback Shift registers (LFSRs). Some transformation method is then applied to convert the uniform distribution into Gaussian distribution.

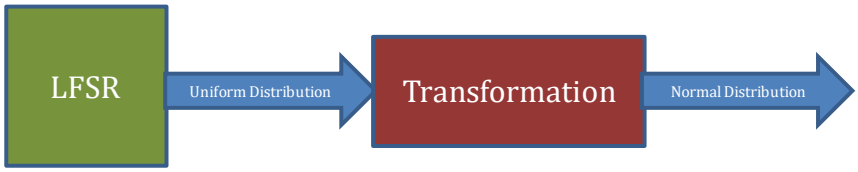

Figure 2.1: *Basic Methodology to Generate GRNs*

Normal distribution is an open-ended distribution in which values of increasing magnitude occur with rare probabilities. *Tail accuracy* indicates the similarity between a given distribution and the ideal normal distribution at high sigma (standard deviation) values. It is one of the most important parameter of the quality of produced GRNs. Figure 2.2, shows the GRNs produced on linear and logarithmic scales. The uniform distribution is also shown with flat density in figure 2.2. A careful observation of the plots shows that the high probability region is more prominent on linear scale while, the less probability region is more prominent on logarithmic scale. In high sigma or standard deviation region, a careful observer can see the irregular shape of PDF on logarithmic scale. This is because of the statistical nature of samples. Due to low probability of occurrence of samples, the tail region in the statistically plotted PDF never becomes smooth.

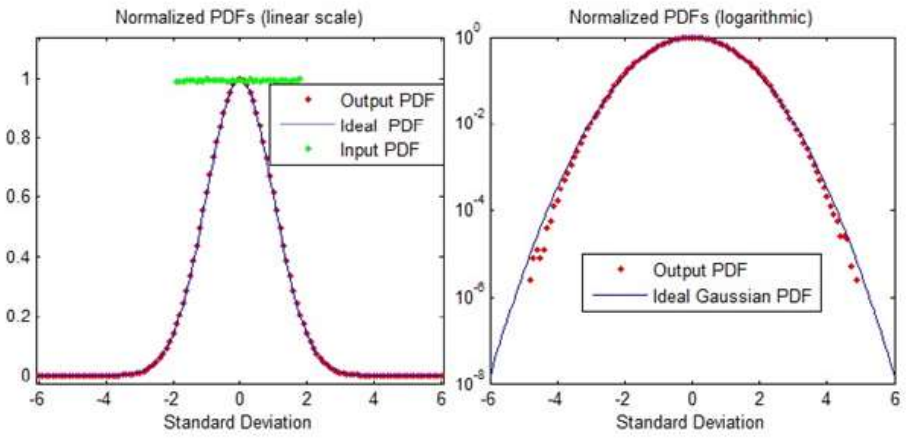

Figure 2.2: *PDF of Gaussian Random Numbers (GRNs), (a) Linear Scale, (b) Log. Scale*

2.2 Classifications of GRNGs

The transformation methods for generating GRNs are mainly classified into two broad categories [3].

1) Exact Methods

The exact methods produce ideal GRNs if the perfect arithmetic is used. For example, the Box Muller transformation algorithm applies various transformation methods on uniform random numbers to produce GRNs. Perfect GRNs will be produced if the uniform random numbers and the arithmetic functions involved in the method are evaluated using infinite arithmetic precision. Another example is Ziggurat algorithm, which also lies under the category of exact methods that produces GRNs of any arbitrary tail accuracy by approximating Inverse Gaussian CDF. Ziggurat algorithm is currently being used to generate Gaussian samples by well known MATLAB software.

2) Approximate Methods

The approximate methods, on the other hand, generate approximate Gaussian samples even if the infinite precision arithmetic is used. For example, basic Central Limit Theorem, that produces approximate Gaussian samples by averaging *n* uniform samples. This method becomes exact when infinite uniform samples are combined.

GRNs generators can also be classified into four basic categories [3];

1) Inversion Methods
2) Transformation Methods
3) Rejection-Acceptance Methods
4) Recursive Methods

Inversion methods include CDF inversion that simply inverts the Gaussian CDF to produce random number from Gaussian distribution. Transformation method includes Box-Muller method that applies some transformation on uniform random numbers to generate Gaussian distributed numbers. Rejection-Acceptance methods include Ziggurat algorithm that produce GRNs based upon conditional rejection acceptance criteria applied to certain transformed values. The recursive methods include Wallace method that produces new Gaussian output samples with the help previously generated Gaussian outputs using a feedback network. In the upcoming subsections, various algorithms for generating GRNs are explained in detail.

2.2.1 The CDF Inversion Method [8]

The CDF inversion method generates a random number from the normal distribution by approximating the Inverse Gaussian Cumulative Distribution Function (IGCDF) described by the following equation;

$$n = F^{-1}(u) \tag{2.1}$$

where, “u” is the number from a uniform distribution, “n” is the number form Gaussian distribution and F^{-1} is the IGCDF as shown in figure 2.3.

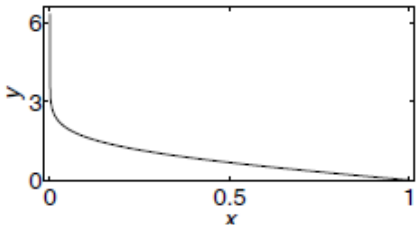

Figure 2.3: *Inverse Gaussian Function ($n = F^{-1}(u)$)*

The conceptual block diagram of the algorithm is given in figure 2.4.

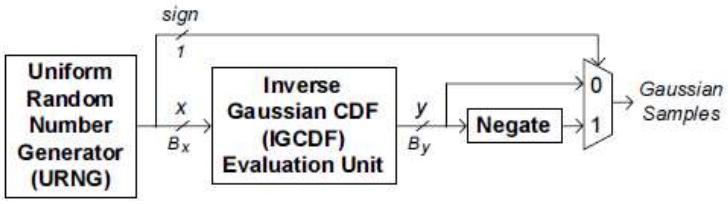

Figure 2.4: *High Level Architecture of CDF Inversion Method*

This method requires a large amount of memory to store the CDF inverse, especially, for the data in tail region of the Gaussian distribution. The most efficient method reported in [8] uses the *hierarchical non-uniform segmentation* algorithm that reduces the large memory requirement. In its segmentation approach, one out of four segmentation schemes (US, $P2S_L$, $P2S_R$, $P2S_{LR}$) is selected for initial segmentation of an input function. US stands for uniform segments. $P2S_L$ implies increasing number of segments with power of two beginning from the start of input range to the end. $P2S_R$ implies decreasing number of segments with power of two beginning from the start of input range till end. $P2S_{LR}$ implies increasing number of segments with power of two till mid of the given input range and then decreasing number of segments with power of two till end. The term *hierarchical* is used because of the fact that in first step, the entire region is divided using one of the above four segmentation schemes in. In the next step, each segment is

further subdivided into inner segments using *US*. A detailed range and precision analysis of the arithmetic used for *hierarchical non-uniform segmentation* scheme is also given in [8]. Table 2.1 summarizes the hardware implementation details of the algorithm.

Table 2.1: *Implementation results of degree-1 and degree-2 spline CDF Inversion based GRNG on a Xilinx Virtex-5 FPGA*

Method	Degree-1	Degree-2
Slices	543	579
Block RAMs	2	1
DSP Slices	2	4
Clock Speed [MHz]	371	370
Samples / Cycle	1	1
Million Samples / Sec	371	370
Throughput / Slice	0.683	0.639

2.2.2 Ziggurat Algorithm [6]

In Ziggurat algorithm, the Gaussian probability density function is partitioned into three different regions to generate Gaussian samples. These are *rectangular, wedge and tail* regions as shown in figure 2.5. A rejection-acceptance criterion is used to examine whether a random sample falls into one of these three regions.

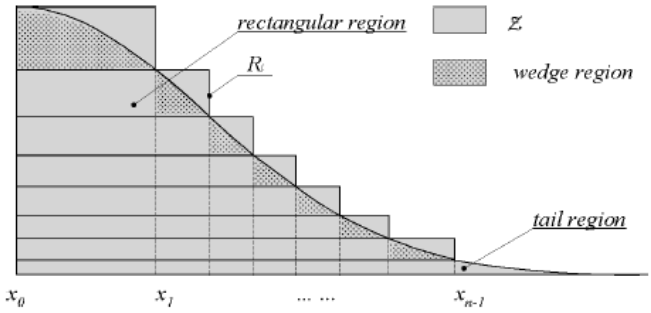

Figure 2.5: *Rectangular, Wedge and Tail Regions in Ziggurat Algorithm*

The highest probability of occurrence of a random sample lies in rectangular region. The numbers falling in this region are directly taken as output random numbers from Gaussian distribution. Secondly, there is a 1.5 percent chance of occurrence of a random input sample in non-rectangular region because it is a low probability region. The exponential or logarithmic

functions are evaluated and an iterative fixed point operation unit is used for the samples occurring in non-rectangular region. The algorithm [3] for ziggurat method is given below;

loop
  $i \leftarrow 1 + \lfloor nU_1 \rfloor$ {Usually n is a binary power: can be done with bitwise mask}
  $x \leftarrow x_i U_2$
  if $|x| < x_{i-1}$ **then**
    return z {Point completely within rectangle.}
  else if $i \neq n$ **then** {Note that $\phi(x_{i-1})$ and $\phi(x_i)$ are table look-ups.}
    $y \leftarrow (\phi(x_{i-1}) - \phi(x_i))U$ {Generate random vertical position.}
    if $y < (\phi(x) - \phi(x_i))$ **then** {Test position against PDF.}
      return x {Point is inside wedge.}
    end if
  else
    return $|x| > r$ from the tail {see section 3}
  end if
end loop

The summary of the latest hardware implementation of ziggurat algorithm is given in Table 2.2. The algorithm provides any arbitrary tail accuracy but suffers from a problem that its output rate is not constant, which means that some of the times samples are missing in continuously running clock cycles. Also, there is a need for more accurate evaluation of complex elementary functions involved in ziggurat method as its future extension.

Table 2.2: *Implementation results for Ziggurat based GRNG on XC2VP30-6 and XC3S200-4 FPGAs*

	XC2VP30-6	XC3S200-4
SLICEs	868 out of 13,696 (6%)	908 out of 1,920 (47%)
Block RAMs	4 out of 136 (2%)	4 out of 12 (33%)
MULT18X18s	2 out of 136 (13%)	2 out of 12 (16%)
DCMs	1 out of 8 (12%)	1 out of 4 (25%)
Period of "CLK"	5.88ns (170MHz)	6.11ns (164MHz)
Period of "CLK2"	11.76ns (85MHz)	12.21ns (82MHz)

2.2.3 Polar Method [7]

The polar method requires two uniform random numbers to covert into two Gaussian output random numbers. The two uniform random numbers are taken between the range -1 and +1 and the magnitude of their vector in polar plane is computed. If the vector magnitude is greater than 1, the numbers are discarded. If magnitude is less than 1, the magnitude of the vector is

transformed and scaled to give two Gaussian output numbers. The algorithm is described as under,

repeat
  $x \leftarrow V_1$, $y \leftarrow V_2$
  $d \leftarrow x^2 + y^2$
until $0 < d < 1$
$f \leftarrow \sqrt{-2(\ln d)/d}$
return $(f \times x, f \times y)$

The algorithm requires the computation of a division, square root and two multiplications. Also, the output rate no longer remains constant due to conditional *if-then-else* statements. The BER simulation result of a communication system [7] over AWGN channel based on polar method is illustrated in figure 2.6.

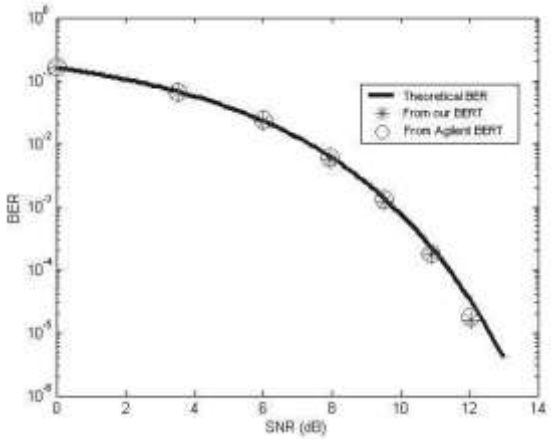

Figure 2.6: *BER Simulation Results of Polar Method*

2.2.4 GRAND Algorithm [3]

The GRAND algorithm, also known as the odd-even method, belongs to the class of exact GRNs generators. It transforms uniform random numbers into Gaussian distributed, by using the formula given by eq.2.2

$$f(x) = ke^{-G(x)} \tag{2.2}$$

where, “x” is a random sample from uniform distribution, “k” is a constant and “G(x)” is a probability density function of any arbitrary distribution and its range is between 0 and 1. For

the Gaussian distribution the G(x) is given by eq.2.3.

$$G(x)=\frac{1}{2}(x^2-a^2) \tag{2.3}$$

In order to keep the range in between 0 and 1, it is necessary to divide the range of distribution into various sections as shown in figure 2.7.

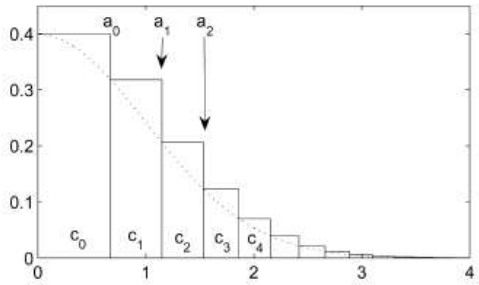

Figure 2.7: *Subsections for GRAND Method*

The algorithm of the GRAND method is given below;

$i \leftarrow 0, \;\; x \leftarrow U$ {Note that $0 < x < 1$ according to definition of U}
while $x < 0.5$ **do** {Generate i with geometric distribution}
  $x \leftarrow 2x, \;\; i \leftarrow i + 1$
end while
loop {Now sample within chosen segment using odd-even method}
  $u \leftarrow (a_{i+1} - a_i)U_1$
  $v \leftarrow u(u/2 + a_i)$
  repeat
    if $v < U_2$ **then**
      if $U_3 < 0.5$ **then**
        return $a_i + u$
      else
        return $-a_i - u$

      end if
    else
      $v \leftarrow U_4$
    end if
  until $v < U_5$
end loop

The output rate of GRAND method is not constant due to conditional *if-then-else* statements as evident from the algorithm given above. Also, the efficient hardware implementation of this

method has not been reported yet.

2.2.5 Wallace Method [9]

The Wallace method eliminates the requirement of evaluation of complex functions like sine, cosine, natural logarithms etc. thus, making it suitable for applications that require high throughput rates. The conceptual block diagram of Wallace method is shown in figure 2.8.

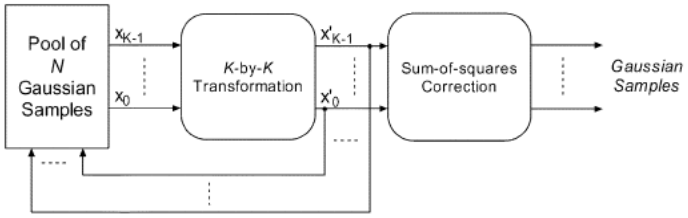

Figure 2.8: *Overview of Wallace Method*

This method applies k-by-k transformation using Hadamard orthogonal matrix [3] on a pool of already generated Gaussian samples "N", to produce new Gaussian outputs. The main drawback associated with this method is its inherent feedback nature that causes severe correlations between successive data samples. Correlation effect is highly undesired characteristic for the Gaussian distributed samples because it indicates the presence of some frequencies or similarities in the resultant distribution. The Wallace method is highly resource consuming but provides a high throughput rate. The algorithm for the Wallace method is given below and its hardware implementation is shown in figure 2.9.

```
for i = 1..R do   {R = retention factor}
  for j = 1..L do   {L = N/K}
    for z = 1..K do   {K = matrix size}
      x[z] ← pool[generate_addr()]
    end for      {Apply matrix transformation to the K values}
    x' ← transform(x)
    for z = 1..K do   {write K values to pool}
      pool[generate_addr()] ← x[z]'
    end for
  end for
end for
S ← √(pool[N]/N)   {Approximate a χ²_N correction for sum of squares.}
return pool[1..(N − 1)] × S   {Return pool with scaled sum of squares.}
```

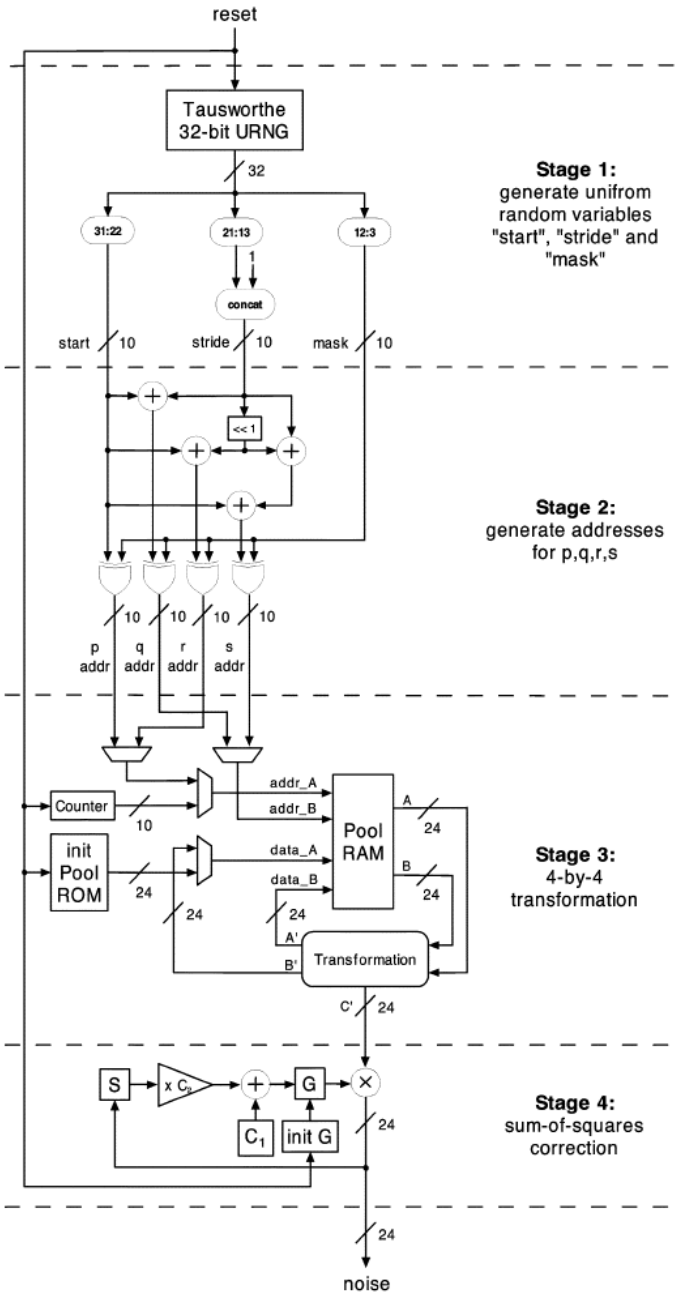

Figure 2.9: *Hardware Architecture of Wallace Method*

The hardware resource utilization is given in Table 2.3.

Table 2.3: *Implementation results for Wallace based GRNGs on XC2V4000-6 FPGA*

stage	slices	block RAMs	MULT18X18s
1	77	-	-
2	121	-	-
3	342	4	-
4	230	2	4
total	770	6	4

2.2.6 Box Muller Transformation Method [2]

This method is the most popular method for generating high quality GRNs in hardware. It falls under the category of exact algorithms. It basically, transforms two input uniform random numbers to provide two Gaussian samples. The Box Muller algorithm is given by below;

$$X_1 = \sqrt{-2\ln U_1}\cos(2\pi U_2) \tag{2.4}$$

$$X_2 = \sqrt{-2\ln U_2}\cos(2\pi U_1) \tag{2.5}$$

The conceptual block diagram of the Box-Muller method is shown in figure 2.10. Various algorithms and techniques have been applied to implement the above mentioned Box-Muller equations in hardware. In [10], Boutillon reported hardware implementation of Gaussian noise generator based on Box-Muller algorithm where CLT is used to reduce the approximation errors of the mathematical functions involved in Box-Muller method. In [11], mathematical functions in Box-Muller method are approximated using degree one piecewise polynomial approximation along with non-uniform segmentation scheme. Due to high approximation and quantization errors, CLT is employed to enhance the noise quality. The output rate is one sample per clock cycle and highest attainable tail accuracy is 6.7σ. Dong-U Lee, in [12], presented an accurate analytical error analysis and bit width optimization for mathematical functions of Box-Muller method. CLT is not used, thus, providing the output rate of two samples per clock cycle. The highest attainable tail accuracy is 8.2σ. This method [12] is highly efficient both in terms of

hardware and throughput compared to previously reported method.

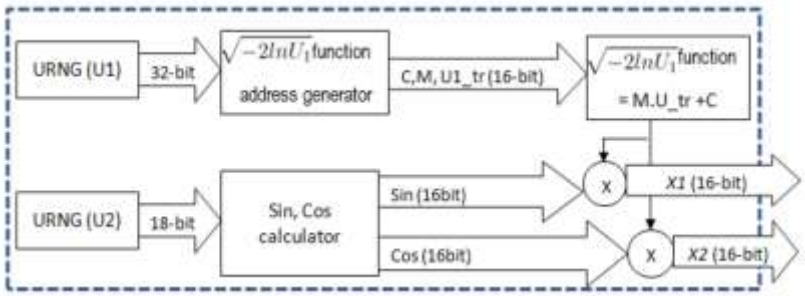

Figure 2.10: Hardware Architecture of Box-Muller Method

Among all these implementations, the most efficient hardware implementation of Box-Muller method is reported in [2], that achieves lower hardware cost and maximum attainable sigma values larger than previously published designs. The method uses polynomial curve fitting with hybrid segmentation and scaling scheme to more accurately approximate the mathematical functions involved in Box-Muller method. The resource utilization summary is given in Table 2.4. The ASIC implementation of the architecture is given in figure 2.11.

The tail accuracy till 9.4 sigma has been shown using PDF plots in [2]. The PDF plot for 10^{11} Gaussian samples till 6σ is shown in figure 2.9. The chi-square simulation results are shown in Table 2.5.

Table 2.4: *Implementation results for BM-Method*

Device[a]	I	II	III	IV
Bitwidth of u_1	32	32	64	32
Period of the PNG	$\approx 2^{113}$	$\approx 2^{88}$	$\approx 2^{258}$	$\approx 2^{133}$
Max. deviation	6.66σ	6.66σ	9.4σ	6.66σ
Clock freq. (MHz)	269	248	248	336
Output rate (MGVs/sec)	538	496	496	672
Number of logic cells	576	534	852	668
Resource utilization	1.3%	2.3%	3.6%	0.4%
On-chip memory blocks	2	2	3	2
18×18*-bit Mults.*	3	3	3	3

[a] Design I was synthesized for a Xilinx Virtex-II Pro XC2VP100-6 FPGA. Designs II and III were synthesized for a Xilinx Virtex-II XC2V4000-6 FPGA. Design IV was synthesized for an Altera StratixII EP2S180F1508C4 FPGA.

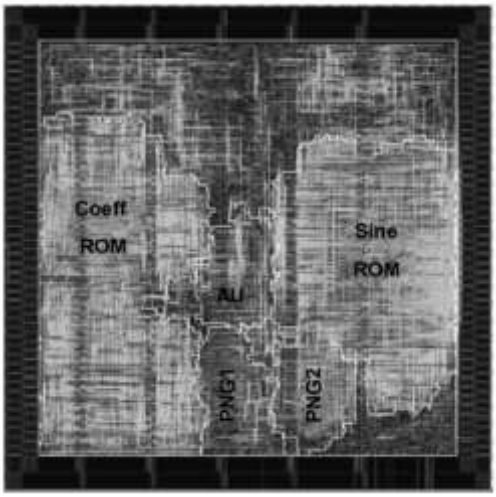

Figure 2.11: *ASIC Implementation of Box-Muller Method*

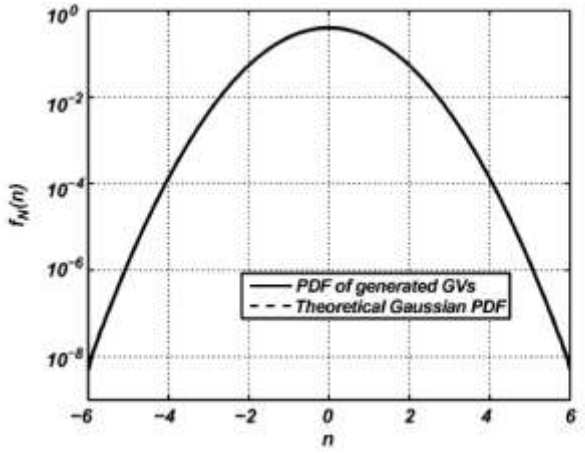

Figure 2.12: *PDF plot of BM Method till 6σ for 10^{11} samples*

Table 2.5: *Chi-Square test results for BM-Method*

Range	*Calculated* $\chi^2_{\alpha,97}$	*Threshold*			
		132	121	115	111
		α 0.01	0.05	0.10	0.15
$\lvert n\rvert \leq 4.5$	109	pass	pass	pass	pass
$4.5 < \lvert n\rvert < 6.0$	90	pass	pass	pass	pass
$6.0 < \lvert n\rvert < 7.5$	103	pass	pass	pass	pass
$7.5 < \lvert n\rvert < 9.4$	110	pass	pass	pass	pass

Xilinx [13] have released an IP core based on Box-Muller algorithm. ASIC chip implementation of Box-Muller Method is given by Fung [14].

2.3 Work Related to Central Limit Theorem [3]

The Central Limit Theorem (CLT) states that the sum of infinitely large uniform random numbers approaches to Gaussian distribution. CLT has been used in various algorithms to improve the noise quality of the generated Gaussian samples. It has been implemented earlier in software using the methods given below;

- In order to enhance the tail accuracy, the idea is to "stretch" the PDF in the tail region
- **Teichroew (1953)** used a Chebyshev interpolating polynomial to map the PDF of CLT, for a given *n*, to that of Gaussian distribution
- **Muller** in **1959** used a 9^{th} degree polynomial on the sum of 12 uniform random numbers
- The degree of polynomials used in the above mentioned techniques increases the Complexity of the algorithm. Therefore, a tradeoff has to be made between the accuracy and complexity. The corrected PDF still deviates from ideal Gaussian PDF for any given value of *n*. Also, averaging large number of uniform random numbers constitute a big computational challenge.
- That is why *Central Limit Theorem was rarely used in hardware implementation of high quality GRNGs*

To date, no hardware implementation of improved Central Limit Theorem has been reported. Hence this work, to the best of our knowledge, is the first attempt to generate high tail accuracy GRNs in hardware using CLT.

CHAPTER 3

A NOVEL ARCHITECTURE OFAWGN GENERATOR

3.1 Ideal Gaussian Distribution

The normal (or Gaussian) distribution is normally used to represent real-valued random variables that tend to accumulate around mean value. The PDF equation of a normal distribution is given as,

$$f(x)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x-\mu)^2/2\sigma^2} \tag{3.1}$$

where, σ^2 is the variance and μ is the mean or expected value of normal distribution. For a *standard normal distribution*, mean is 0 and variance is 1. The PDF equation becomes

$$f(x)=\frac{1}{\sqrt{2\pi}}e^{-x^2/2} \tag{3.2}$$

Gaussian distribution does not deviate from its mean by more than 3 standard deviations 99.7 percent of the times or in other words, the probability of occurrence of data samples in high probability region is very low [2] as shown in figure 3.1.

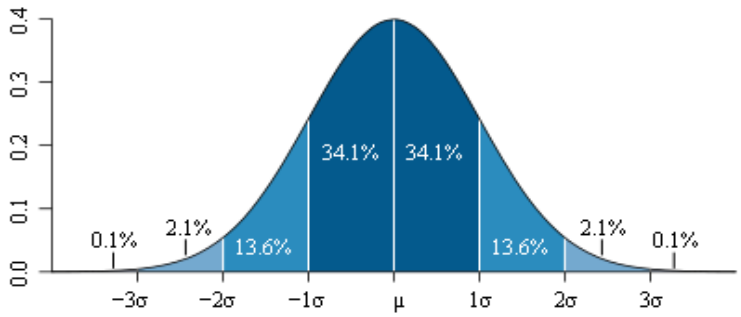

Figure 3.1: *Ideal Gaussian Distribution (Image courtesy: Wikipedia)*

3.2 White Noise

White noise is a random signal as shown in figure 3.2 with a flat Power Spectral Density (PSD) or in other words, whose response is a low pass filter effect [17]. A perfect white noise is independent and identically distributed, which implies no autocorrelation (correlation between successive samples) as shown in figure 3.3. If a white noise signal is normally distributed with mean zero and variance σ^2, then, it is called as the *Gaussian white noise signal or Uniform random sequence.*

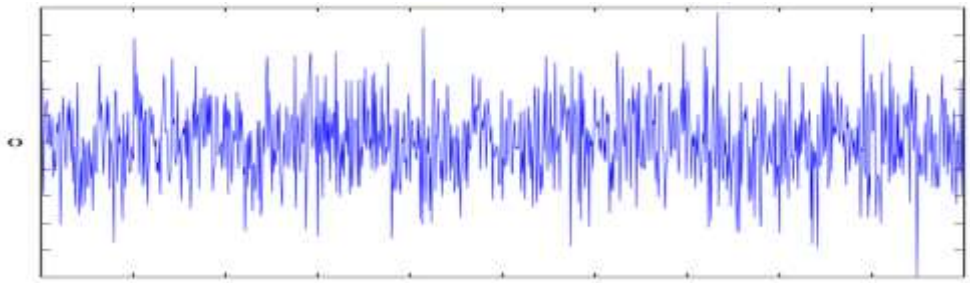

Figure 3.2: *White Noise Process*

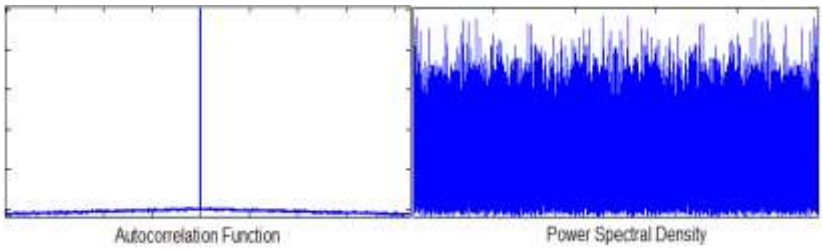

Figure 3.3: *Autocorrelation and Power Spectral Density (PSD) of White Noise*

3.3 Central Limit Theorem (CLT)

The sum of *n* independent random variables, each with finite mean and variance, becomes Gaussian distributed by virtue of CLT [1]. As *n* increases, resultant distribution becomes closer to Gaussian. The normalized random variable is given as

$$Z_n = \frac{\sum_{i=1}^{n} X_i - \sum_{i=1}^{n} \mu_i}{\sum_{i=1}^{n} \sigma_i} \tag{3.3}$$

where, X_i is the i[th] independent random variable, μ_i is the mean or expected value of the i[th] independent random variable and σ_i is the standard deviation of the i[th] independent random variable.

If n approaches infinity or under certain regularity conditions, the limiting distribution of Z_n is standard normal distribution.

$$\lim_{n\to\infty} F_{Z_n}(t) = \lim_{n\to\infty} \Pr\{Z_n \le t)\} = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy \tag{3.4}$$

$$Z_n \to N(0,1) \tag{3.5}$$

where, $N(0,1)$ is the normal distribution whose mean is 0 and standard deviation is 1.

CLT is extremely simple and efficient in implementation as it produces GRNs by simply adding *n* numbers with arbitrary PDFs. But, this method has never been used to generate high quality Gaussian samples. This is due to error in tail region of the probability density function (PDF), which is inversely proportional to number of samples to be added. For example, addition of at least 100 samples is required to achieve a tail accuracy of 5σ. The PDF curve deviates from the ideal Gaussian PDF very sharply for higher values of standard deviation and also, the PDF tends to become straight in tail region. To achieve a better tail accuracy, the value of *n (number of*

additions) needs to be increased. In fact, in order to produce a perfect PDF, n should be infinitely large.

This is shown in figure 3.4 where normalized PDF is plotted for varying values of *n.* Base numbers or initial numbers are uniformly distributed between +1 and -1. We generated 10^9 samples by adding 2, 4, 8 and 16 uniformly distributed numbers respectively. In Figure 3.4a, the PDF for the generated samples is plotted on linear scale. Apparently, the PDF seems close to ideal Gaussian PDF for values of *n* greater than 2. This is because for standard deviation greater than 2σ, the probability becomes too small to be detected on linear scale. In Figure 3.4b, the PDF is plotted on logarithmic scale. Now, the errors in tail region are more prominent. It can be clearly seen that even for *n*= 16, PDF deviates from ideal curve drastically after 3σ. It is, also, apparent that as *n* becomes greater, PDF curves for CLT tend to be closer to ideal Gaussian curve.

A careful observer can notice irregular shape of PDF at the edges particularly for higher values of n. This is due to the statistical nature of data samples. No matter how many data samples are taken, edge (tail) of a statistically plotted PDF is never smooth due to low probability of occurrence of data samples in this region as explained in section 3.1.

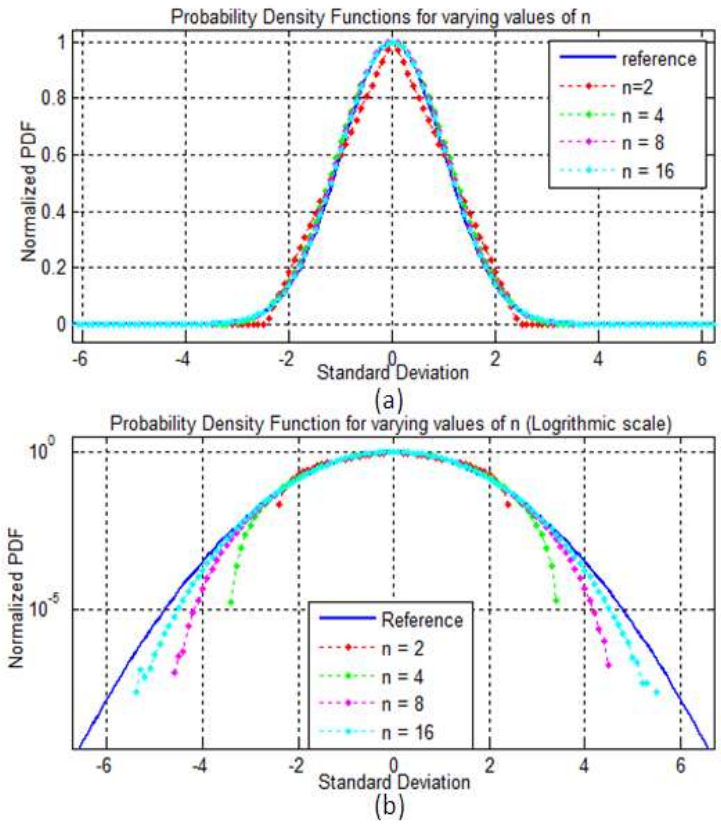

Figure 3.4: *PDF for varying values of n, (a) Linear Scale, (b) Logarithmic Scale*

3.4 Calculating Variances for Central Limit Theorem

As shown in figure 3.4 that PDF for a specified value of *n* remains fixed. So, the variance and standard deviation for that specified value of *n* is also fixed. Variance and standard deviation are calculated below for $n = 8$ when the base numbers are normally distributed between +1 and -1. Similar procedure can be followed to find the variances and standard deviations for other values of *n* as summarized in Table 3.1

Table 3.1: *Standard deviations and variances for varying n*

n	Standard Deviation σ	Variance σ^2
2	0.816496581	0.666666667
4	1.154700538	1.333333333
8	1.632993162	2.666666667
16	2.3094010773	5.333333333
32	3.26598632	10.66666666

The PDF of a uniform distribution is given as,

$$f_U(x) = \frac{1}{b-a} \quad \text{if } a<x<b, \text{ else } 0 \tag{3.6}$$

The expected value or mean value is given as,

$$E(X) = \mu_X = \frac{a+b}{2} \tag{3.7}$$

The variance is given as,

$$Var(X) = \sigma_X^2 = \frac{(b-a)^2}{12} \tag{3.8}$$

If lower limit a = -1 and upper limit b = +1, then mean value is calculated as,

$$E(X) = \mu_X = \frac{a+b}{2} = \frac{-1+1}{2} = 0 \tag{3.9}$$

Variance is computed as,

$$Var(X) = \sigma_X^2 = \frac{(b-a)^2}{12} = \frac{(1+1)^2}{12} = \frac{1}{3} \tag{3.10}$$

And the standard deviation evaluates to,

$$Std(X) = \sqrt{\frac{1}{3}} \tag{3.11}$$

Let "Y" be a random variable that represents the summation of *eight* uniform random numbers between +1 and -1. X_i is the i[th] uniform distribution. Then variance is calculated as,

$$Y = X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + X_8 = \sum_{i=1}^{8} X_i \tag{3.12}$$

$$E(Y) = \mu_Y = \sum_{i=1}^{8} E(X_i) = 8*0 = 0 \tag{3.13}$$

$$Var(Y) = \sigma_Y^2 = \sum_{i=1}^{8} Var(X_i) = 8*\frac{1}{3} = \frac{8}{3} = 2.667 \tag{3.14}$$

$$Std(Y) = \sigma_Y = \sqrt{\frac{8}{3}} = 1.632 \tag{3.15}$$

3.5 Computing Error in Central Limit Theorem

The variance and standard deviation for any value of *n* in CLT is fixed as shown in section 3.4. Also, the PDF curves for both the CLT and ideal Gaussian distribution remain fixed for any value *n*. So, we can map the CLT curve on to the ideal one by following the concept described in the next subsection.

3.5.1 The Idea

Analyzing figure 3.4 reveals that in the high probability region before 3 sigma, both the curves, the CLT and the reference, are indistinguishable (overlapping each other) and after 3 sigma both the curves grow in different directions. For simplicity, we call the overlapping region as the ***Ideal*** region and non-overlapping region as the ***Non-Ideal*** region.

Therefore, we can say that,

"When in Non-Ideal region, Scale up the random number"

Now, the “n” degree polynomial is given as

$$y = a_n P^n + a_{n-1} P^{n-1} + \ldots + a_2 P^2 + a_1 P + a_0 \quad (3.16)$$

where, a_n ….. a_0 are the polynomial constants.

Reducing this equation to only first degree, we get

$$y = a_1 P + a_0 \quad (3.17)$$

This is equivalent to

$$y = MP + C \quad (3.18)$$

where, “M” is the slope and “C” is the intercept of a straight line. This means that we can scale-up a number P by simply multiplying it with a slope and adding an intercept value.

3.5.2 Algorithm for Error Computation

The error function for CLT can be computed empirically by using the algorithm explained below;

1- Uniform random numbers are generated between +1 and -1. A MATLAB script is written to generate random numbers as

```
urng = 2*(rand(1,M)-0.5)
```

where, “rand” is a built in function to generate a uniform random signal between 0 and 1 and “M” defines the length of the generated random signal.

For a specified value of *n*, such *n* uniform random distributions are generated and added by virtue of Central Limit Theorem. The resultant distribution is of length “M” and is called as the approximate Gaussian distribution. Clearly, the highest value sample will be +*n* and the lowest value sample will be –*n*.

2- To compute the PDF of generated approximate Gaussian samples, the samples are

distributed into buckets or bins over the sigma or standard deviation range from -*n* to +*n* as described in step 1. The results are shown in figure 4.4 for varying values of *n*. The values corresponding to the sigma axis or horizontal axis are represented as x_c

3- We have the PDF of an ideal Gaussian distribution with zero mean as given by eq. 3.19;

$$f(x_c) = \frac{1}{\sqrt{2\pi}} e^{-x_g^2/2\sigma_n^2} \tag{3.19}$$

where, x_g are the points on horizontal axis corresponding to ideal Gaussian distribution. Now, the sigma or standard deviation values corresponding to ideal Gaussian distribution can be found using the PDF values for approximate Gaussian distribution by rearranging eq. 3.19. The standard deviation in terms of PDF is given by the following equation;

$$x_g = \sqrt{-2\sigma_n^2 \ln(\sqrt{2\pi\sigma_n^2} f(x_c))} \tag{3.20}$$

where, $f(x_c)$ is the probability of occurrence of x_c

4- The error in the CLT PDF is then simply;

$$Err_{PDF} = x_c - x_g \tag{3.21}$$

The plots in figure 3.5 and figure 3.6 are obtained for the positive side with the symmetric negative side, using the above algorithm for n = 4 and 8. For each point on the CLT curve, there is a corresponding straight or horizontal point on the reference curve.

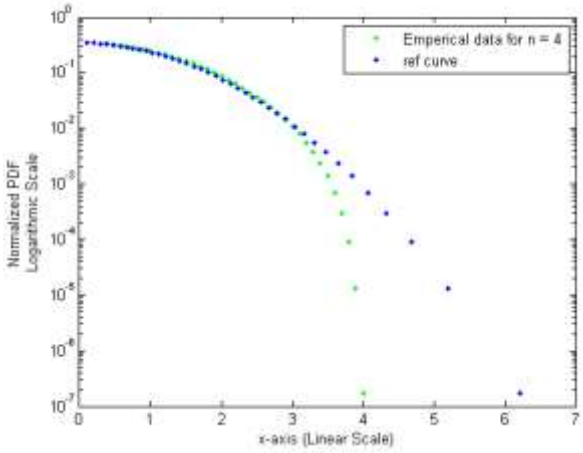

Figure 3.5: *PDF Plot for n = 4*

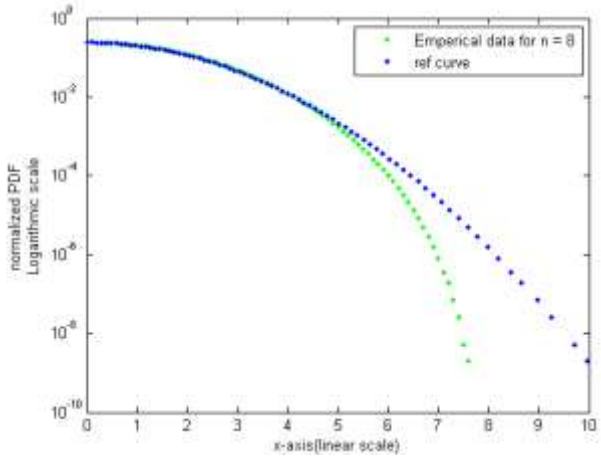

Figure 3.6: *PDF Plot for n = 8*

3.5.3 Error Distributions

The above algorithm was implemented in MATLAB script for varying values of n. In Figure 3.7, the computed error functions are shown. The horizontal axis has been normalized to standard normal distribution using fixed variances for CLT PDFs.

It is evident from figure 3.7 that there is an initial error growing towards the positive side and

after that, the curve decays exponentially. This means that the positive error region requires negative polynomial constants and vice versa.

There is, also, a statistical inaccuracy in the tail region which is because of the statistical nature of the data samples. The error goes on decreasing as *n* increases, because the higher value of *n* takes the curve closer to the ideal reference curve.

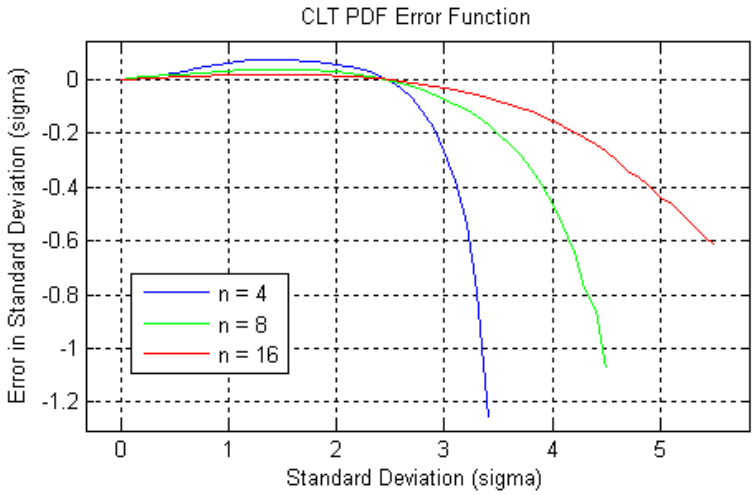

Figure 3.7: *Error in PDF for Varying n*

3.6 Compensating Error in Central Limit Theorem

This subsection explains the methodology to compensate and correct the above computed error distribution. Methodology for $n = 8$ and a 16 bit datapath will be explained that provides the tail accuracy of 6σ. However, depending upon available computational resources and desired tail accuracy, this methodology can be used for any value of *n* and bit width.

Now, the accuracy of the algorithm depends upon how the best we model this error function so that it can be implemented using minimal hardware resources.

As it has been mentioned before, our idea is to use first degree polynomial as it leads to the most efficient implementation of the algorithm in hardware. So, first degree piecewise polynomial approximation will be used to model the error distribution.

3.6.1 Uniform Segmentation Algorithm

Piecewise polynomial approximation is applied using uniform segmentation algorithm. The entire error curve has been divided into 16 uniform segments. The figure 3.8 shows the approximation technique applied and the residual error

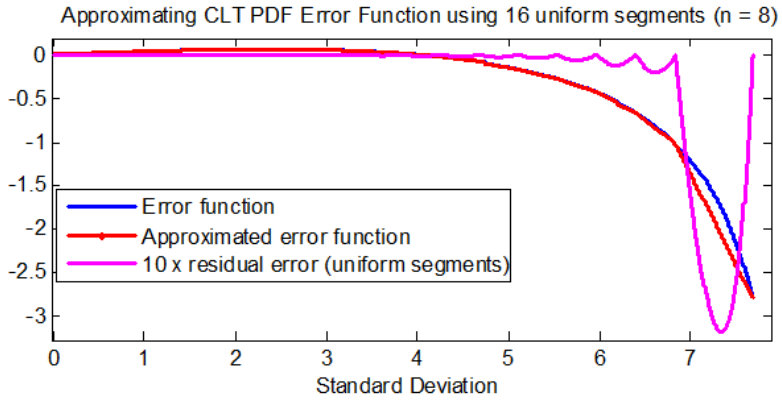

Figure 3.8: *Residual Error due to Uniform Segmentation*

The residual error is observed to be growing exponentially with increasing standard deviation. This is because the error function is a *non-linear function*. Hence, uniform segmentation is not suitable for such functions. The problem is to find a solution so that,

$$Err_{max} - Err_{min} \rightarrow 0 \qquad (3.22)$$

where, Err_{max} is the maximum residual error and Err_{min} is the minimum residual error.

3.6.2 A Novel Non-Uniform Segmentation Algorithm

A simple and efficient algorithm is proposed to compute non-uniform segments with minimum residual error as described below;

The algorithm exploits the fact that residual error due to piecewise polynomial approximation of non-linear function is directly proportional to *curvature* of the function [16].

The curvature defines the deviation of a non-linear curve along its path. Let us suppose that we have a position vector $r(t) = \langle x(t), y(t) \rangle$ that follows a path in the plane with the passage of

time as shown in the figure 3.9**.** We know that the velocity is the instantaneous rate at which the position vector changes with respect to time. So that $r'(t) = \langle x'(t), y'(t) \rangle$ and the unit tangent vector is shown in figure 3.9.

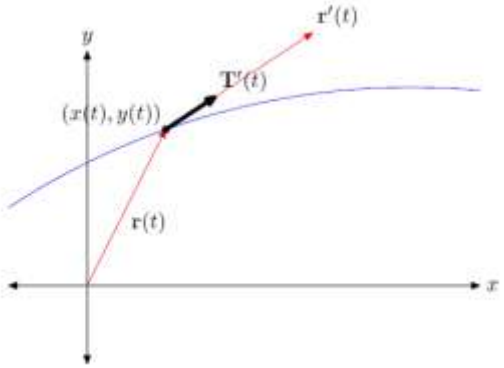

Figure 3.9: *The unit tangent vector T'(t), points in the direction of velocity vector*

If the unit tangent vector is changing rapidly with respect to time (dT'/dt), then there is a great deal of deviation of the curve at that point. If the unit vector is changing slowly with time, it means that there is not a great deal of deviation in the curve at that point.

Obviously, the derivative of the unit tangent vector has a close relationship with the *curvature* of a non-linear function. So, we define curvature of function *k* with respect to rate of change of its unit vector *T* (also called as the velocity vector) as

$$k = dT/ds \tag{3.23}$$

Or simply, curvature of a non-linear function can be defined as its second order derivative as shown in figure 3.11.

Once the curvature of a function is known, the problem is reduced to finding the segments on the curve with length of each segment proportional to the curvature of the function.

These sample points or segments can be found using Amplitude to Frequency Converter (AFC). A simple AFC (designed in simulink, MATLAB) is shown in figure 3.10.

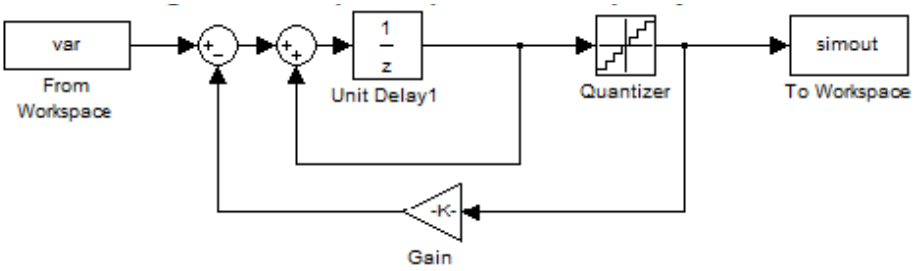

Figure 3.10: *Amplitude to Frequency (AFC) Converter*

This is basically a Sigma-Delta modulator that modulates a carrier signal with respect to the input signal amplitude. A feedback gain "K" is provided to control the oscillations at the output of AFC. Output of the AFC is the sampling or frequency points on the input function with respect to the change in amplitude of the function.

3.6.3 Residual Error

Second order derivative of the error distribution is given as an input to AFC and the output is a discrete signal whose frequency is proportional to the input signal. By changing the gain on the feedback path, we can tune the AFC to provide any number of sampling points. This is shown in figure 3.11**.** We have tuned the AFC to provide 17 sampling points (or 16 segments) on the entire error function.

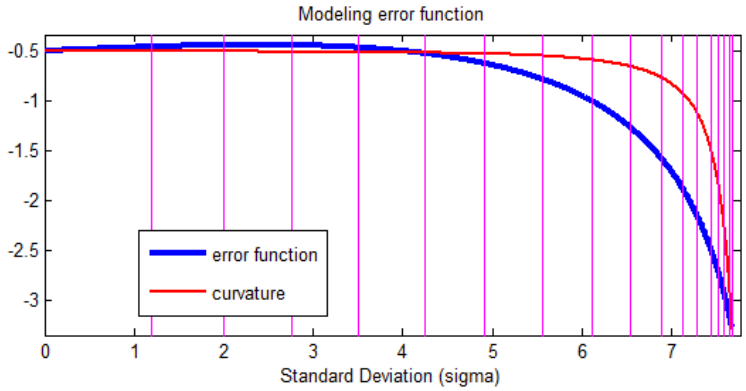

Figure 3.11: *Non-Uniform Segmentation Scheme*

Now, using the known non-uniform segments, we can apply the piecewise polynomial approximation to the error function and find the residual error**.** Clearly, the non-uniform segmentation drastically reduces the residual error as shown in figure 3.12.

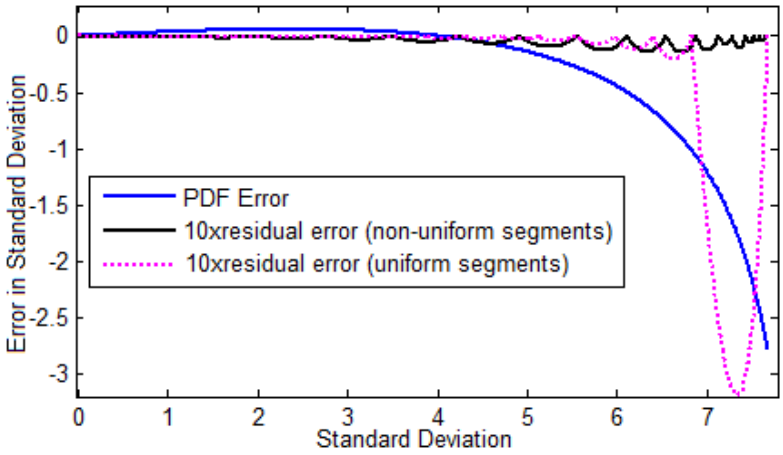

Figure 3.12: *Residual Error due to Non-Uniform Segmentation Scheme*

3.7 Hardware Implementation

The algorithm described in the previous section is implemented in hardware using verilog HDL and fixed point arithmetic. The data path is of 16 bit and calculations are performed for n = 8. The input is in Q1.15 format and the output is in Q4.12 format. The design is partitioned into four distinct blocks, Linear Feedback Shift Registers (LFSRs), summation block, decision block and first order polynomial calculator, and is explained in detail in upcoming sub-sections.

3.7.1 Uniform Random Number Generator

As stated earlier, that Uniform Random Numbers (URNGs) are also known as white noise, which essentially covers all frequency spectrum range. In hardware, Linear Feedback Shift Registers (LFSRs) are commonly utilized as noise and data sources because they are easily implemented and require minimal hardware resources [17].

Hardware implementation of an LFSR (flip flop or storage registers) and a feedback network as shown in figure 3.13. The generator polynomial of LFSR is implemented by the feedback path as

given by eq. 3.16. The generator polynomial describes the statistical characteristics and sequence length (period) of LFSR. If the sequence period of LFSR is 2^{n-1}, then the sequence is known as the *maximal length sequence.*

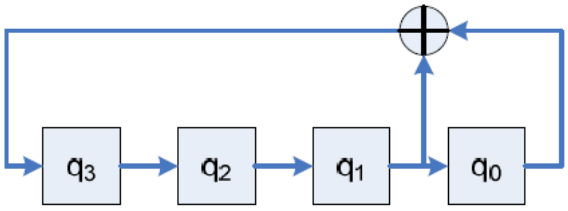

Figure 3.13: *Basic LFSR Architecture*

However, this class of pseudo random number generators (PNGs) shares a common disadvantage that they exhibit severe correlations between successive data samples as shown in figure 3.14. This effect can be seen in covariance and power spectral density plots shown in figures 3.14 and 3.15 respectively. The covariance is not a delta function, as would be expected for uncorrelated values, but, it shows significant correlations up to +8 and -8 lags. Also, the power spectral density is not flat as expected for white noise (section 3.2), but the low pass effect is clearly seen in the figure 3.15.

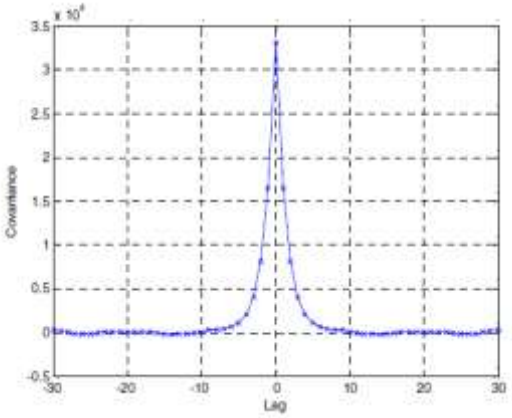

Figure 3.14: *Covariance of 10^5 values generated by LFSR using maximal length polynomial shows correlations around zero lag*

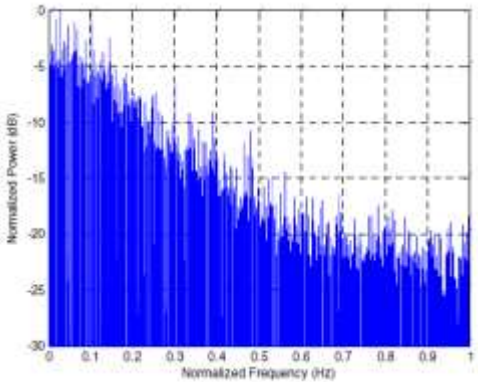

Figure 3.15: *Power Spectral Density of 10^5 Values Generated by LFSR*

The correlation effect can be understood by observing that, for any step in the sequence, the generating polynomial changes only a few of the bits in registers (flops) [17].

The 2-D scatter plot in figure 3.16 clearly shows a visible lattice structure that indicates the correlation between adjacent data samples [2].

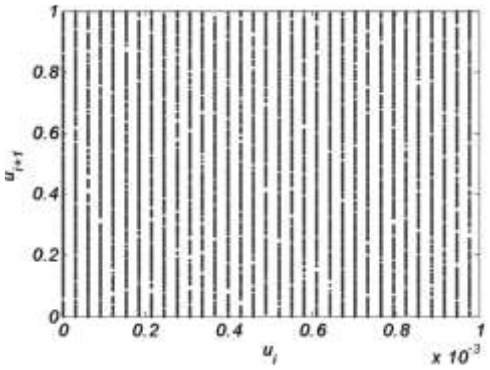

Figure 3.16: *2-D scatter plot of a pseudo random number generator*

There are many techniques to overcome these short comings. A simple approach is to advance the LFSR by number of steps “k” so that the required number of bits has changed before the arrival of new output. This method proves to be costly because the LFSR has to be executed “k” times faster than rest of the design.

In this thesis work, we have used Skip-Ahead LFSR logic that follows from the algorithm presented by Leonard Calvito [17]. It is the most recent and simplest technique which can

advance the LFSR to “k” steps ahead in a single step and, also, requires minimal hardware resources.

Our requirement is to use eight 16-bit LFSRs as base numbers. So, we designed a 128-bit Skip Ahead LFSR architecture using maximal length polynomial. This technique basically uses a transition matrix “A^k” which can advance the LFSR to k steps ahead. LFSR can be represented by the eq. 3.24;

$$q^{(t+k)} = A^k q^{(t)} \tag{3.24}$$

where,

$$\mathbf{A} = \begin{bmatrix} a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \\ 1 & 0 & & & 0 \\ 0 & 1 & & & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{q} = \begin{bmatrix} q_{n-1} \\ q_{n-2} \\ \vdots \\ q_1 \\ q_0 \end{bmatrix}$$

In our design, we advanced this LFSR by k = 16 to ensure that successive samples are un-correlated. Eight 16-bit LFSRs are taken from the 128-bit Skip Ahead design. The results are shown in figure 3.17. The output is taken in Q1.15 format for each 16 bit Uniform Random Number Generator (URNG).

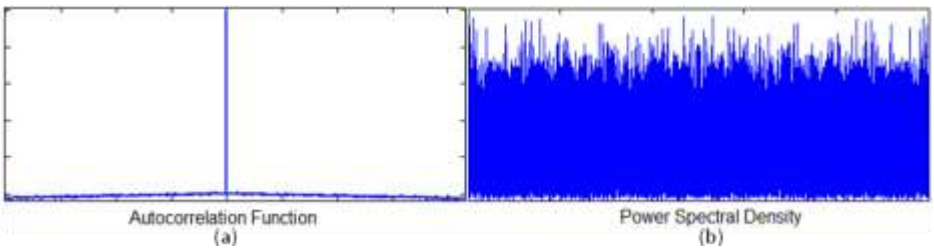

Figure 3.17: *(a) Autocorrelation Function (ACF), (b) PSD Plots for Skip Ahead LFSR.*

3.7.2 Summation Block

The summation block can also be called as averaging block as it implements the Central Limit Theorem. This block basically implements a hierarchical structure of seven 16-bit adders as

shown in figure 3.18. To perform the averaging process in hardware, we discard 3 least significant bits from the output of this block which is equivalent to dividing the summation of eight numbers by 8 (2^3). So, at each clock edge, 8 URNGs with range +1 and -1 are simply added and divided by 8 simultaneously. The final output of this block is in Q4.12 format within the range of +8 and -8. The architecture is fully pipelined and symmetric. In the intermediate stages, as indicated, we get the Gaussian samples at lower sigma accuracies and the final output gives the accuracy up to 3σ.

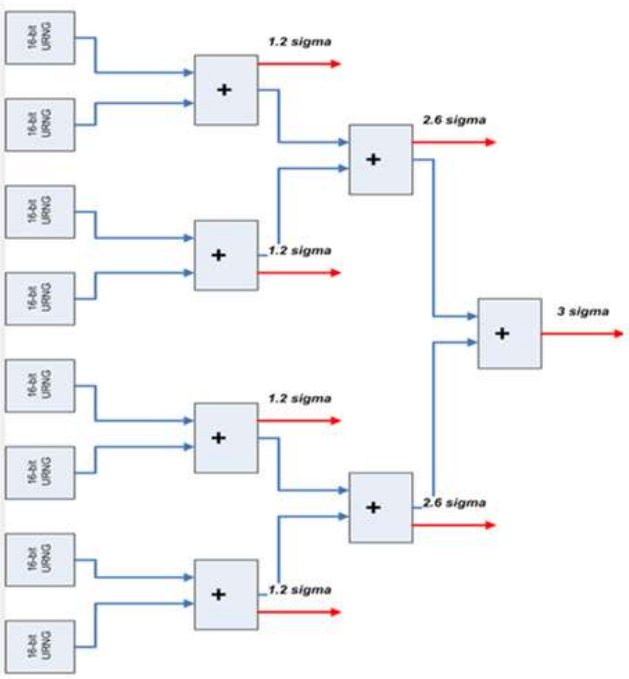

Figure 3.18: *Summation Block Architecture*

3.7.3 Decision Block

The decision block or compensation block is the most important and novel part of the hardware GRN generator. The input of this block is a 16 bit number in Q4.12 format from the summation block. It comprises of two look-up-tables (LUTs) which contain coefficients to be used with the first order polynomial calculator. An address generator selects the coefficients from LUTs based upon input number from summation block. The *search space* is made hierarchical to make the

output rate constant as shown in figure 3.19. That is why the number of segments chosen was 16 so that we could a symmetric architecture in powers of two. Based upon 16 non-uniform segments computed using the algorithm described in section 3.63, coefficients a_0 and b_0 are pre-calculated and stored in the LUTs. The whole search space architecture is shown in figure 3.19 within the range of *0 to 7.649* sigma.

As indicated in table 3.1, standard deviation for n = 8 is 1.632. Hence, to get the Gaussian samples at the sigma or standard deviation scale or normalized variance, every output needs to be divided by 1.632. This will require an additional multiplier in the architecture which proves to be costly. However, this multiplier can be avoided by *pre-dividing* the Look-up-Table (LUT) coefficients a_0 and b_0. We call these as transformed coefficients and denote them by a_{tr} and b_{tr} respectively. Hence,

$$a_{tr} = \frac{a_0}{\sigma_8} \text{.....} b_{tr} = \frac{b_0}{\sigma_8} \tag{3.25}$$

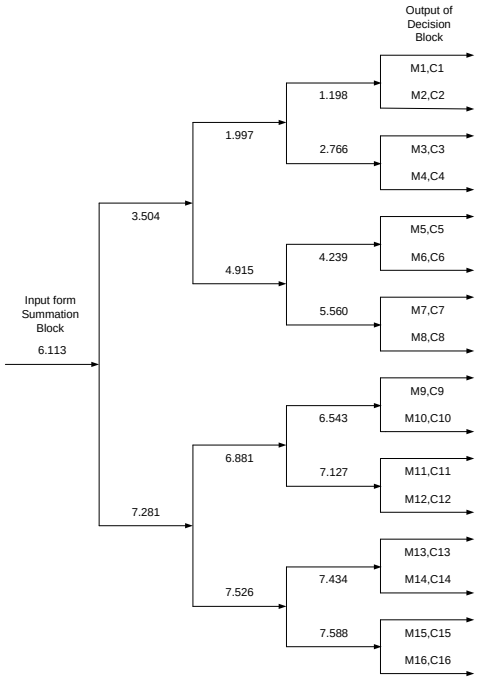

Figure 3.19: *Decision Block Architecture*

3.7.4 Contents of Look-up-Tables (LUTs)

The pre-computed values of a_{tr} and b_{tr} for n = 8 using 16 non-uniform segments are shown in Table 4.2. Here we can decide the bit widths of the coefficients in fixed in arithmetic. From the table 4.2, it can be noticed that the highest absolute value of a_{tr} is 2.98586 and the highest absolute value of b_{tr} is 16.0741. So, we take Q2.14 format for a_{tr} and Q4.12 format for b_{tr}.

Table 3.2: *Pre-computed values of a_{tr} and b_{tr}*

Segment	a_{tr}	b_{tr}	Segment	a_{tr}	b_{tr}
1	0.60311	0	**9**	1.00640	-1.98295
2	0.61158	-0.00994	**10**	1.19128	-3.16846
3	0.62492	-0.03604	**11**	1.42539	-4.74712
4	0.64493	-0.09029	**12**	1.67874	-6.51661
5	0.67398	-0.19003	**13**	1.97865	-8.65662
6	0.71590	-0.36419	**14**	2.64596	-11.2220
7	0.77646	-0.65589	**15**	2.33078	-13.5465
8	0.86953	-1.16301	**16**	2.98586	-16.0741

3.7.5 Polynomial Calculator

Polynomial calculator is the final block of the GRNs generator. It comprises of an adder and a multiplier. The decision block provides the random number "*x*" and its corresponding coefficients a_{tr} and b_{tr}. The multiplier result is truncated to fit into 16 bits as per rules of fixed point arithmetic. The output of this block is a Gaussian random number in Q4.12 format. Corrected Gaussian number x_{cor} is then generated by first degree polynomial equation;

$$x_{cor} = a_{tr}x + b_{tr} \tag{3.26}$$

The proposed GRNs hardware architecture is shown in figure 3.20.

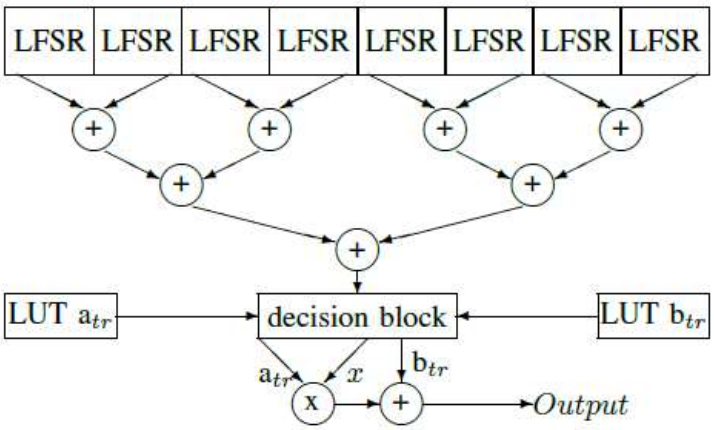

Figure 3.20: *Architecture of Proposed GRNs Generator*

The base numbers (LFSR) are uniformly distributed within 0 and 1. The pseudo-code for the proposed improved CLT architecture, for any value of *n*, is given below,

$$x_1 \leftarrow U_1, x_2 \leftarrow U_2, x_3 \leftarrow U_3, ..., x_n \leftarrow U_n$$
$$x \leftarrow \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$
$$x \leftarrow \frac{x - mean(x)}{std(x)}$$
$$x \leftarrow select(m, c)$$
$$return(x_{cor} = mx + c)$$

The proposed architecture is implemented on FPGA (Virtex-4) using Verilog HDL. The results obtained are explained and discussed in detail in the next chapter.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Results

Since, the error function estimation in our methodology is based upon empirical data analysis, it does not guarantee arbitrarily accurate Gaussian numbers. However, after correction, the architecture should provide accurate Gaussian numbers with tail accuracy till 6σ. To, validate this claim, we have applied standard tests to measure statistical accuracy of generated numbers. These tests include Probability Density Function (PDF) plots, Chi-Square tests, Scatter plots and Autocorrelation function.

4.2 PDF Plots

To obtain tail accuracy till 6σ, at least 10^{11} samples are required [2]. At such high sigma values the probability is too low to observe on linear scale. The architecture explained in chapter 3 is used to generate 10^{11} Gaussian samples. Figure 4.1 shows the PDF plots of the generated GRNs on logarithmic scale as compared to ideal Gaussian PDF as well as PDF of random numbers generated by simple summation (CLT).

Although, accuracy till 6σ is guaranteed, the algorithm is capable of giving even higher sigma accuracy (till 7σ) as described in chapter 3. To achieve this, the number of generated Gaussian samples should be increased. Since, the length of segments decrease with increasing sigma values, hence, the curve becomes more accurate and smooth at the low probability or tail region (non-uniform segmentation).

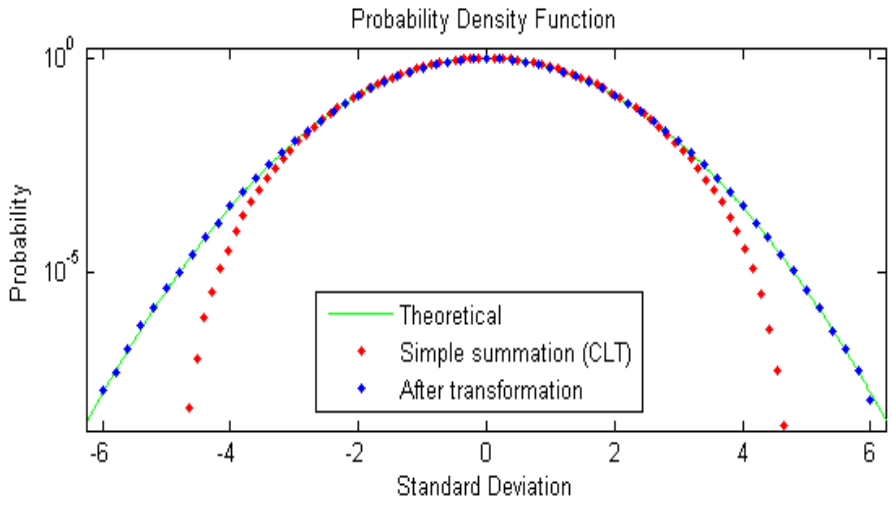

Figure 4.1: *PDF of Proposed GRNs Generator*

4.3 Statistical Goodness-of-Fit Test

A Chi-Square test is called as the statistical goodness-of-fit test. It is a statistical hypothesis test in which the sampling distribution is said to be a Chi-Square distribution when the null hypothesis is true. Both the standard and tail generation algorithms are evaluated using chi-square test [3]. This test is normally used to verify the normality of generated Gaussian samples. A set of observed samples (observed frequencies) is compared against the expected distribution (expected frequencies). Using more bins or buckets gives higher resolution with respect to the different input values, but reduces the expected number in each bin [3]. The test is performed by dividing the data samples into number of bins. For each bin, observed and expected counts are calculated. The Chi-Square test statistic is then computed using the formula;

$$\chi^2_{\alpha,\gamma-1} = \sum_{i=1}^{\gamma} \frac{[O(i)-E(i)]^2}{E(i)} \tag{4.1}$$

$$E(i) = KP_i \tag{4.2}$$

where, 'O (.)' is the observed counts, 'E (.)' is the expected counts, 'K' is the number of generated Gaussian variates, 'KP_i' is the expected number of samples according to normal

distribution [1], 'α' is the desired significance level, 'γ' is the total number of bins in which the distribution is divided into and '$\gamma-1$' is the degree of freedom. Since, the normal distribution is completely characterized by two parameters, the mean and the standard deviation, the degree of freedom is thus reduced by 2 from $\gamma-1$ to $\gamma-3$.

Chi-square test is performed by dividing the horizontal axis (sigma axis) into three regions from 0σ to 6σ as shown in Table 4.1. Each interval is segmented into 100, 50 and 30 bins respectively. In order to eliminate the statistical inaccuracy, it has been ensured that at least 50 samples should fall into each bin. Chi-Square test is pass for a given value of α, if the observed value (calculated by eq.4.1) is less than or equal to the corresponding theoretical value.

The significance level α means rejecting the null hypothesis when actually it is true or accepting the null hypothesis when actually it is false. These both criteria's are satisfied if the number of samples in the bins are large enough (statistical accuracy) in the bin whose chi-square value is going to be calculated.

The results obtained, as shown in Table 4.1, indicate that the proposed GRN generator successfully passes the chi square over the entire range of 0σ to 6σ within 5 percent of significance level α.

Table 4.1: *Chi-Square test results for proposed algorithm*

Range	χ^2_{obs}	$\alpha = 0.1$	$\alpha = .05$	$\alpha = .01$
$0 \leq \sigma \leq 3.0$ *(100-bins)*	33	114 *(pass)*	121 *(pass)*	132 *(pass)*
$3.0 \leq \sigma \leq 4.5$ *(50-bins)*	53	60 *(pass)*	64 *(pass)*	73 *(pass)*
$4.5 \leq \sigma \leq 6.0$ *(30-bins)*	38	38 *(pass)*	40 *(pass)*	47 *(pass)*

It is worth-mentioning here that, since, our method of GRNs generation is not exact, it will always have error in the PDF as shown in figure 3.12. Hence, as the number of samples increase, the numerator term in eq.4.1 increases as square of the value leading to greater chi-square value.

However, as mentioned earlier, we have used at least 50 samples in every bin for the tests in Table 4.1 and the test passes for the 10 percent or less significance. This is even better than the generally accepted criteria of 5 percent or less significance level [18].

4.4 Scatter Plot and Autocorrelation Test

Any correlation between neighboring numbers can be seen as a regular lattice structure as shown in figure 3.16. This is an unwanted property in any random sequence as it indicates some similarities between adjacent data samples that leads to the undesired low pass characteristics as explained in chapter 3. Figure 4.2 shows a 2-D scatter plot of generated GRNs with no visible lattice structure. The plot indicates that most of the times the samples tend to cluster around the mean value and with very less probability they occur in high sigma regions. This can, also, be seen in figure 4.3 which shows the Autocorrelation function over a range of ± 2000 lags. Correlation values for all non-zero lags are extremely low.

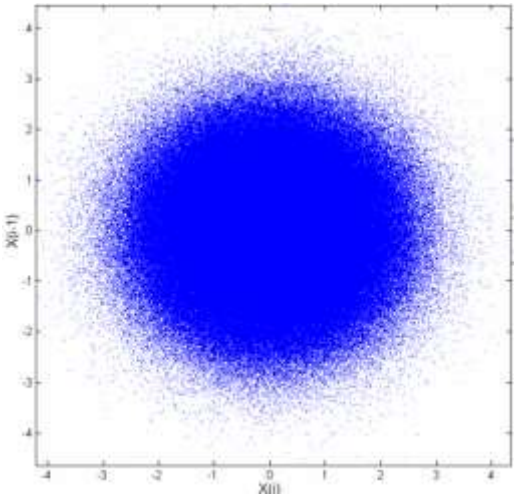

Figure 4.2: *2-D Scatter Plot of Proposed GRNs Generator*

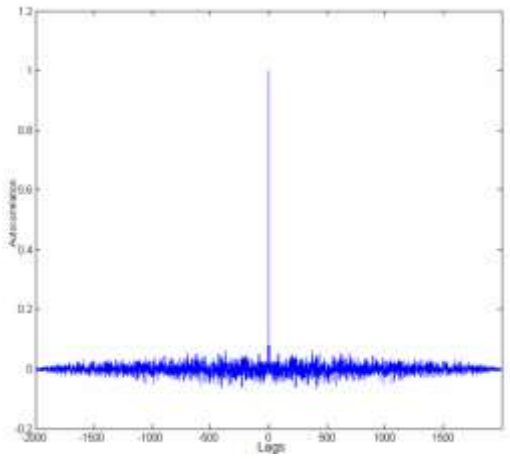

Figure 4.3: *Autocorrelation Function (ACF) of Proposed GRNs Generator*

4.5 BER Simulation

The proposed architecture of improved CLT has been used in a communication system with BPSK signaling over AWGN channel. In figure 4.4, the Bit-Error-Rate (BER) simulation for basic CLT shows the incorrect results and shifts in Signal-to-Noise Ratio (SNR) values at low BER values. In figure 4.5, the BER simulation has been done using our AWGN generator taking the signal length of 10^7. The results are as accurate as for BM method and show no shift in SNR at low BER values. Hence, inaccurate GRNs generator may lead to incorrect simulation results.

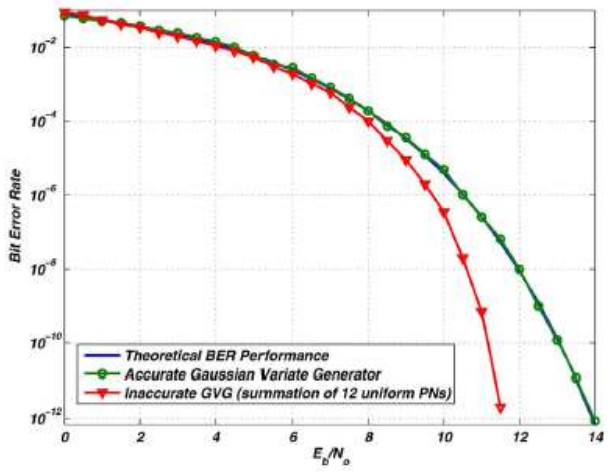

Figure 4.4: *BER simulation of BPSK modulated communication system [2]*

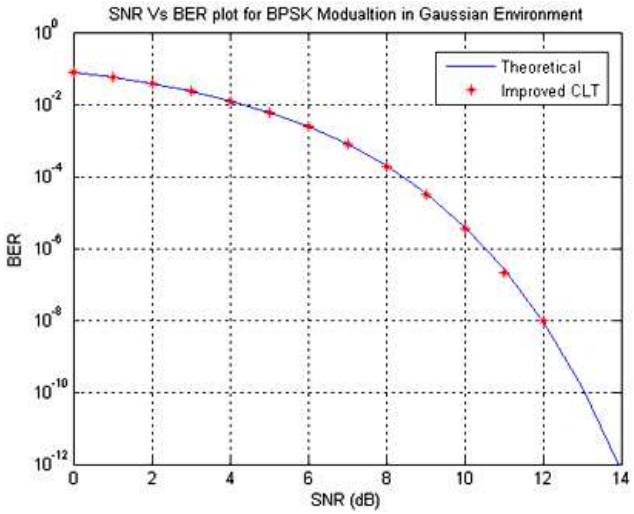

Figure 4.5: *Proposed GRNs Generator Applied in BER Simulation of BPSK Modulated Communication System*

4.6 Synthesis Results

The proposed GRNs generator architecture is implemented in Verilog HDL. The design was synthesized using an older FPGA device family (Xilinx XC4VLX15 Virtex-4 device). This is because, later versions of FPGAs (for example, Virtex-5 and Virtex-6) are faster and have more resources available in a single configurable slice. Hence, for the sake of fair comparison with the previously reported work, we used FPGA from an older family.

The design requires 440 configurable slices and 1 DSP block. No memory blocks are utilized because the memory for LUTs and pipeline registers has been implemented within the used configurable blocks. The design speed (Mega Samples/sec.) is 220MHz with critical path delay of 4.54 ms.

4.7 Comparison with Previous Methods

A detailed comparison of our proposed architecture is given in Table 4.2 with well-known previously reported architectures.

Table 4.2: *Comparison of improved CLT with published work*

Design	[10]	[14]	[2]	[12]	[9]	[6]	This work
Method Used	BM	BM	BM	BM	Wlc.	Zgrt.	**CLT**
Logic Cells	437	480	534	1528	770	891	**420**
Memory Blocks	0.5	5	2	3	6	2	**0**
Multi-pliers	N/A	5	3	12	4	2	**1**
GRN Bitwidth	12	16	16	16	24	32	**16**
Tail Accuracy	4σ	4.8σ	6.6σ	8σ	7σ	N/A	6σ
Speed (M samples/s)	25	245	440	468	155	168	**220**

The comparison has been done in terms of logic cells utilized, memory blocks, number of multipliers used, bit width of GRNG, speed and tail accuracy. The first four implementations are the most widely used architectures of Box-Muller method. The BM method is also being used by the Xilinx core [14]. Ziggurat algorithm is being used in MATLAB.

In terms of configurable logic cells utilization, our design is much better than other designs, since, it requires only 440 slices. BM method requires 2 memory blocks and 3 multipliers to achieve a tail accuracy of 6.6σ, while our design achieves the closer accuracy by using only one multiplier and no memory block. The BM method produces two Gaussian samples per clock cycle so, its speed is twice than our architecture (440). The ziggurat algorithm is quite efficient in implementation and provides any arbitrary tail accuracy (N/A). But, as explained earlier, that its output rate is not constant. It means that sometimes the samples are missing on some edges of clock cycles. This may lead to inaccurate simulation results. Wallace method is not efficient in terms of hardware resources and, also, has a drawback of correlations between successive samples.

Therefore, resource utilization is better than any of the previously reported hardware implementation of GRNs generator. It is worth mentioning here, that our design is scalable to

achieve even higher sigma accuracy with minimal increase in hardware cost. Also, the speed of the design can, also, be increased by using the concept explained in chapter 3 where we are getting Gaussian samples in the intermediate stages with lower tail accuracies.

CHAPTER 5

CONCLUSION

5.1 Conclusions

This work achieves high tail accuracy GRNs generator. The empirical model of the error in CLT is compensated through deployment of a low cost compensation block. After a detailed error analysis, coefficients for degree one piecewise polynomial approximation have been computed using a novel non-uniform segmentation algorithm.

- The proposed GRNs generator falls under the category of *approximate algorithms.*
- The proposed architecture is highly efficient *(area, speed and hardware cost),* simple, fast, compact and regular as compared to all previous architectures.
- The architecture is *fully pipelined* with an initial delay of 4 clock cycles and thereafter, generates Gaussian samples at every clock edge (constant output rate).
- The proposed GRNs generator successfully passes the chi-square statistical test over the entire range of 0σ to 6σ within 5 percent of significance level α.
- Although, tail accuracy of 6σ is guaranteed, the architecture is *scalable* to achieve even higher tail accuracies with minimal increase in hardware resources.
- The proposed architecture outperforms any previously reported designs.

5.2 Future Extensions

As stated earlier that, the hardware architecture explained in chapter 3, guarantees tail accuracy till 6σ, which is good enough for all practical purposes and the most of the AWGN simulations. This architecture can be further improved in various dimensions as explained below;

- ***Approximation errors*** in PDF can be further reduced using higher order polynomials. Degree one polynomial is used in our design as a simplest method for implementation. Higher order polynomials will require more number of multiplications. Also, the approximation errors can be reduced using higher number of segments. As explained earlier that number of segments will be increased by powers of two. Hence, more memory will be required to store the additional polynomial coefficients (LUTs).

- ***Tail accuracy*** can be improved by increasing number of addition operations (value of *n*) by virtue of CLT. Increasing value of *n* will move the CLT curve closer to the reference and, hence, greater standard deviation or sigma value will be obtained for the highest generated sample value. The tail accuracy can, also, be improved by increasing the bit width of the data path. As stated earlier, we have used 16 bit datapath. Increasing width of the datapath will reduce the quantization errors.

- ***Complexity*** can be further reduced by replacing the multiplication operation with a shift and add operations.

All of the above mentioned improvements involve more hardware resources. Therefore, depending upon the user demand, there will be a trade-off between complexity and accuracy.

REFERENCES

1. H. Fischer, "**A History of the Central Limit Theorem: From Classical to Modern Probability Theory"**, *(2010), Springer. ISBN0387878564.*

2. A. Alimohammad, S. F. Fard, B. F. Cockburn and C. Schlegel, **"A Compact and Accurate Gaussian Variate Generator"**, in *IEEE Transaction on very Large Scale* Integration *(VLSI) Systems, Vol. 16, No. 5, May 2008.*

3. D. B. Thomas , W. Luk, Philip H.W. Leong and J. D. Villasenor**, "Gaussian Random Number Generators",** *in ACM Computing Surveys, Vol. 39, No. 4, Article 11, Publication date: October 2007.*

4. C. Iskander, "**A MATLAB-based Object-Oriented Approach to Multipath Fading Channel Simulation**", *MATLAB White Paper.*

5. Muhammad Ali Shami, Ahmed Hemani, "**Partially Reconfigurable Interconnection Network for Dynamically Reprogrammable Resource Array**", *in IEEE ASICON 2009, 8th International Conference on ASICs.*

6. G. Zhang, P. H. W. Leong, D. Lee, J. D. Villasenor, R. C. C. Cheung, and W. Luk, **"Ziggurat-based hardware Gaussian random number generator,"** in *Proc. IEEE Int. Conf. Field Program. Logic It's Appl., 2005.*

7. Y. Fan, Z. Zilic, M. W. Chiang, "**A versatile high speed bit error rate testing scheme** ", *in Proc. IEEE Int. Symp. Quality Electron. Dec. 2004, pp. 395-400.*

8. D. Lee, W. Luk, Ray C.C. Cheung, J. D. Villasenor, W. Luk, "**Inversion-Based Hardware Gaussian Random Number Generator: A Case Study of Function Evaluation via Hierarchical Segmentation",** *in Field Programmable Technology*

(FTP), IEEE International Conference on Dec. 2006.

9. D. Lee et al., **"A hardware Gaussian noise generator using the Wallace method,"** *in IEEE Transactions on VLSI Systems. Oct. 2007.*

10. E. Boutillon, J. L. Danger, and A. Gazel**, "Design of high speed AWGN communication channel emulator,"** *Analog Integr. Circuits Signal Process, pp. 133–142, 2003.*

11. D. W. Luk, J. D. Villasenor, and P. Y.K. Cheung**, "A Gaussian Noise Generator for Hardware-Based Simulations",** *in IEEE Transactions on Computers, VOL. 53, NO. 12. Dec. 2004.*

12. D.-U. Lee, J. D. Villasenor,W. Luk, and P. H. W. Leong, "**A hardware Gaussian noise generator using the Box–Muller method and its error analysis,"** *IEEE Trans. on Computers, vol. 55, no. 6, pp. 659–671, Jun. 2006.*

13. E. Fung, K. Leung, N. Parimi, M. Purnaprajna, V. Gaudet, "ASIC Implementation of a High Speed WGNG for Communication Channel Emulation ", *Proc. IEEE Workshop Signal Processing Systems, pp. 304-409, 2004.*

14. **"Additive White Gaussian Noise (AWGN) Core",** *v1.0, Xilinx Inc., 2002.*

15. M. E. Muller, **"A comparison of methods for generating normal deviates on digital computers"**, *in Association for Computing Machinery, pp. 376-383, 1959.*

16. D. Arnold, **"Curvature in Matlab ",** *Math 50C Multivariable Calculus.*

17. L.Colavito and D. Silage, **"Efficient PGA LFSR Implementation Whitens Pseudorandom Numbers"**, *in International Conference on Reconfigurable Computing and FPGAs, 2009.*

18. D'Agostino and Stephens, "**Goodness-of-Fit Techniques",** *New York: Marcel Dekker, 1986.*

19. C. M. Grinstead and J. L. Snell, **"Introduction to Probability",** *in American Mathematical Society, pp. 299-301.*